FACULTY OF COMPUTING
SEMESTER II, SESSION 2024/2025

BACHELOR OF COMPUTER SCIENCE (BIOINFORMATICS)

SECB3203 PROGRAMMING FOR BIOINFORMATICS - SECTION 01

PROJECT
ALZHEIMER'S DISEASE PREDICTION

GROUP MEMBERS	MATRIC NO
TAN ZHAO HONG	A23CS0188
CHIN PEI WEN	A23CS0065
KOO XUAN	A23CS0300

LECTURER'S NAME : DR. SEAH CHOON SEN

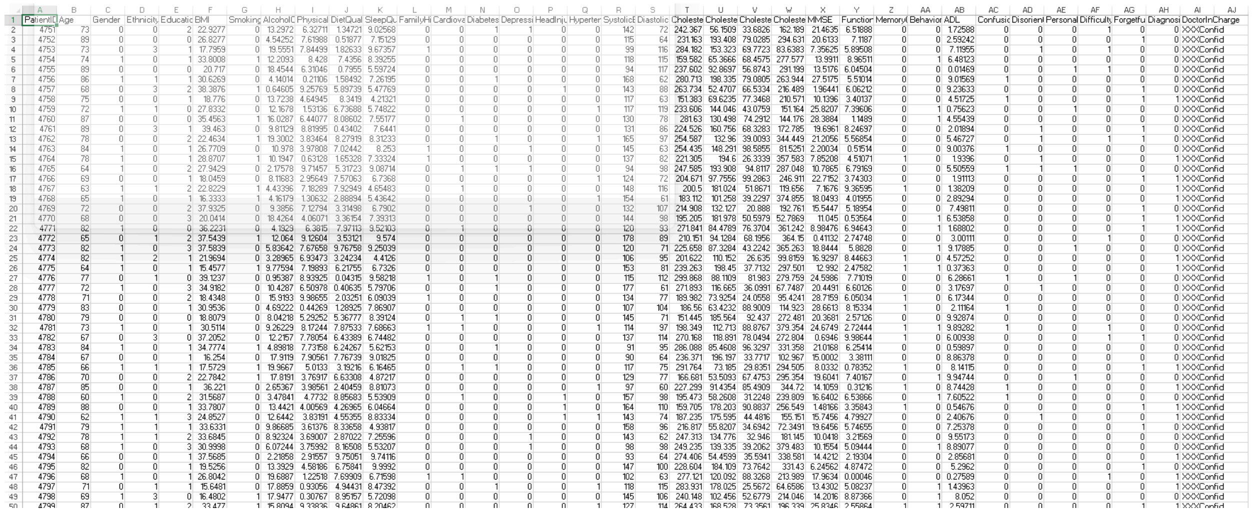
SUBMISSION DATE : 03 DECEMBER 2025

Table of Contents

2.0 Data Collection and Pre-processing

Our dataset was fetched from Kaggle:
https://www.kaggle.com/datasets/rabieelkharoua/alzheimers-disease-dataset/data

The dataset is in .xlsx, thus we convert it to csv file so that it can be easily read by Python.



2.1 Importing Dataset

- Python packages for Data Science

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report,
                             roc_auc_score, roc_curve)
# from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')
```

- pandas: Data manipulation and analysis
- numpy: Numerical operations and array handling
- matplotlib: Data visualization
- seaborn: Statistical data visualization
- scikit-learn: Machine learning algorithms and preprocessing

- Importing Dataset File

```python
# ============================================================================
# 1. DATA LOADING
# ============================================================================

def load_data(filepath):
    """Load the Alzheimer's disease dataset"""
    df = pd.read_csv(filepath)
    print(f"Dataset shape: {df.shape}")
    print(df)
```

```python
if __name__ == "__main__":
    FILEPATH = "alzheimers_disease_data.csv"
    TARGET_COLUMN = 'Diagnosis'
```

```
================================================================================
ALZHEIMER'S DISEASE PREDICTION - ML PIPELINE
================================================================================

[1] Loading Data...
Dataset shape: (2149, 35)
      PatientID  Age  Gender  Ethnicity  EducationLevel        BMI  ...  Disorientation  PersonalityChanges  DifficultyCompletingTasks  Forgetfulness  Diagnosis  DoctorInCharge
0          4751   73       0          0               2  22.927749  ...               0                   0                          1              0          0       XXXConfid
1          4752   89       0          0               0  26.827681  ...               0                   0                          0              1          0       XXXConfid
2          4753   73       0          3               1  17.795882  ...               1                   0                          1              0          0       XXXConfid
3          4754   74       1          0               1  33.800817  ...               0                   0                          0              0          0       XXXConfid
4          4755   89       0          0               0  20.716974  ...               0                   1                          1              0          0       XXXConfid
...         ...  ...     ...        ...             ...        ...  ...             ...                 ...                        ...            ...        ...             ...
2144       6895   61       0          0               1  39.121757  ...               0                   0                          0              0          1       XXXConfid
2145       6896   75       0          0               2  17.857903  ...               0                   0                          0              0          1       XXXConfid
2146       6897   77       0          0               1  15.476479  ...               0                   0                          0              0          1       XXXConfid
2147       6898   78       1          3               1  15.299911  ...               0                   0                          0              1          1       XXXConfid
2148       6899   72       0          0               2  33.289738  ...               1                   0                          0              1          0       XXXConfid

[2149 rows x 35 columns]
```

These screenshots show the initial dataset loading process.

- Dataset Information

```
print(f"\nDataset info:")
print(df.info())
```

```
Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2149 entries, 0 to 2148
Data columns (total 35 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   PatientID                  2149 non-null   int64
 1   Age                        2149 non-null   int64
 2   Gender                     2149 non-null   int64
 3   Ethnicity                  2149 non-null   int64
 4   EducationLevel             2149 non-null   int64
 5   BMI                        2149 non-null   float64
 6   Smoking                    2149 non-null   int64
 7   AlcoholConsumption         2149 non-null   float64
 8   PhysicalActivity           2149 non-null   float64
 9   DietQuality                2149 non-null   float64
 10  SleepQuality               2149 non-null   float64
 11  FamilyHistoryAlzheimers    2149 non-null   int64
 12  CardiovascularDisease      2149 non-null   int64
 13  Diabetes                   2149 non-null   int64
 14  Depression                 2149 non-null   int64
 15  HeadInjury                 2149 non-null   int64
 16  Hypertension               2149 non-null   int64
 17  SystolicBP                 2149 non-null   int64
 18  DiastolicBP                2149 non-null   int64
 19  CholesterolTotal           2149 non-null   float64
 20  CholesterolLDL             2149 non-null   float64
```

```
 21  CholesterolHDL             2149 non-null   float64
 22  CholesterolTriglycerides   2149 non-null   float64
 23  MMSE                       2149 non-null   float64
 24  FunctionalAssessment       2149 non-null   float64
 25  MemoryComplaints           2149 non-null   int64
 26  BehavioralProblems         2149 non-null   int64
 27  ADL                        2149 non-null   float64
 28  Confusion                  2149 non-null   int64
 29  Disorientation             2149 non-null   int64
 30  PersonalityChanges         2149 non-null   int64
 31  DifficultyCompletingTasks  2149 non-null   int64
 32  Forgetfulness              2149 non-null   int64
 33  Diagnosis                  2149 non-null   int64
 34  DoctorInCharge             2149 non-null   object
dtypes: float64(12), int64(22), object(1)
memory usage: 587.7+ KB
None
```

These screenshots show the dataset information details.

- Feature Removal

```python
# ============================================================================
# FEATURES TO REMOVE (Non-medical and weak evidence features)
# ============================================================================
FEATURES_TO_REMOVE = [
    'PatientID',           # Administrative identifier
    'DoctorInCharge',      # Administrative data
    'Gender',              # Weak predictor
    'Ethnicity',           # May introduce bias, weak medical relevance
    'AlcoholConsumption',  # Inconsistent evidence
    'SleepQuality'         # Weak/inconsistent evidence
]

```

```python
    # Remove non-medical and weak evidence features
    print(f"\n{'='*80}")
    print("REMOVING NON-MEDICAL AND WEAK EVIDENCE FEATURES")
    print(f"{'='*80}")
    features_removed = [col for col in FEATURES_TO_REMOVE if col in df.columns]
    if features_removed:
        print(f"\nRemoving features: {features_removed}")
        df = df.drop(columns=features_removed)
        print(f"New dataset shape: {df.shape}")
    else:
        print("\nNo features to remove (features not found in dataset)")

    return df
```

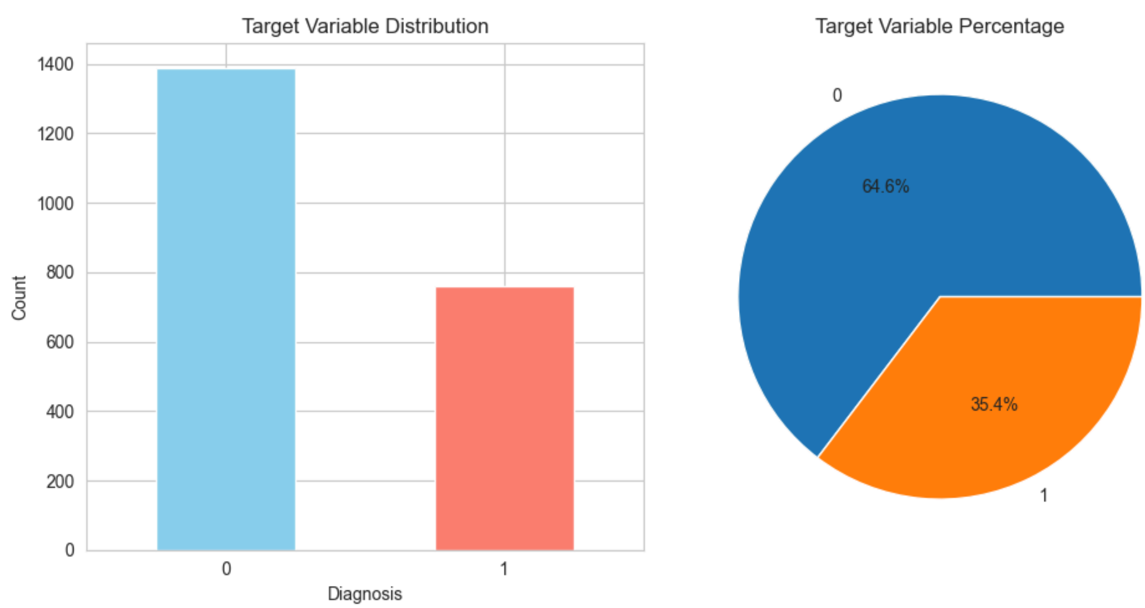
```
================================================================================
REMOVING NON-MEDICAL AND WEAK EVIDENCE FEATURES
================================================================================

Removing features: ['PatientID', 'DoctorInCharge', 'Gender', 'Ethnicity', 'AlcoholConsumption', 'SleepQuality']
New dataset shape: (2149, 29)
```

These screenshots show the removal of non-medical and weak-evidence features such as patient identifiers and administrative attributes. These features were excluded to reduce noise, prevent bias, and ensure that the model focuses only on medically relevant variables that contribute meaningfully to Alzheimer’s disease prediction.

2.2 Data wrangling

- Target Variable Distribution

```python
def preprocess_data(df, target_column='Diagnosis'):
    """Preprocess the data for machine learning"""

    # Separate features and target
    X = df.drop(columns=[target_column])
    y = df[target_column]

    # Encode target variable if it's categorical
    if y.dtype == 'object':
        print(f"\nEncoding target variable")
        le_target = LabelEncoder()
        y = le_target.fit_transform(y)
```

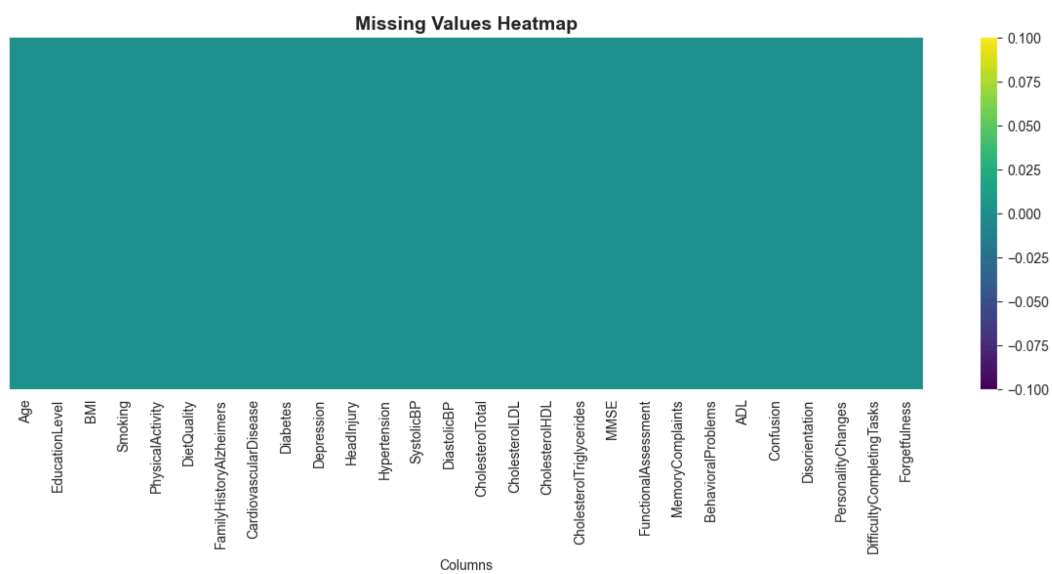


This figure illustrates the distribution of the target variable (Diagnosis) using both bar and pie charts. Visualizing the class distribution helps identify potential class imbalance and confirms that the problem is a binary classification task, which is suitable for the selected machine learning models.

- Identifying and handling missing values

```python
# VISUALIZE MISSING VALUES
print("\n--- Identifying and Handling Missing Values ---")
print("\n1. Missing Values Count:")
print(X.isnull().sum())

print("\n2. Visualizing Missing Values:")
plt.figure(figsize=(12, 6))
sns.heatmap(X.isnull(), cbar=True, cmap='viridis', yticklabels=False)
plt.title('Missing Values Heatmap', fontsize=14, fontweight='bold')
plt.xlabel('Columns')
plt.tight_layout()
plt.savefig('missing_values_heatmap.png', dpi=300)
plt.show()

# Handle missing values
if X.isnull().sum().sum() > 0:
    print("\n3. Handling missing values...")
    X = X.fillna(X.median())
    print("Missing values after handling:")
    print(X.isnull().sum())
else:
    print("\n3. No missing values detected!")
```

```
2. Visualizing Missing Values:

3. No missing values detected!
```



These screenshots display the detection of missing values across all input features. Median imputation was applied to numerical attributes to handle missing data, as it is robust to outliers and preserves the overall distribution of the dataset. This step ensures data completeness before further processing.

- Data Formatting

```python
    # DATA FORMATTING
    print("\n--- Data Formatting ---")
    print("\n1. Current Data Types:")
    print(X.dtypes)

    print("\n2. Numerical Columns:")
    numerical_cols = X.select_dtypes(include=[np.number]).columns
    print(list(numerical_cols))

    print("\n3. Categorical Columns:")
    categorical_cols_orig = X.select_dtypes(include=['object']).columns
    print(list(categorical_cols_orig))
```

```
--- Data Formatting ---

1. Current Data Types:
Age                          int64
EducationLevel               int64
BMI                        float64
Smoking                      int64
PhysicalActivity           float64
DietQuality                float64
FamilyHistoryAlzheimers      int64
CardiovascularDisease        int64
Diabetes                     int64
Depression                   int64
HeadInjury                   int64
Hypertension                 int64
SystolicBP                   int64
DiastolicBP                  int64
CholesterolTotal           float64
CholesterolLDL             float64
CholesterolHDL             float64
CholesterolTriglycerides   float64
MMSE                       float64
FunctionalAssessment       float64
MemoryComplaints             int64
BehavioralProblems           int64
ADL                        float64
Confusion                    int64
Disorientation               int64
PersonalityChanges           int64
DifficultyCompletingTasks    int64
Forgetfulness                int64
dtype: object

2. Numerical Columns:
['Age', 'EducationLevel', 'BMI', 'Smoking', 'PhysicalActivity', 'DietQuality', 'FamilyHistoryAlzheimers', 'CardiovascularDisease', 'Diabetes', 'Depression', 'HeadInjury', 'Hypert
ension', 'SystolicBP', 'DiastolicBP', 'CholesterolTotal', 'CholesterolLDL', 'CholesterolHDL', 'CholesterolTriglycerides', 'MMSE', 'FunctionalAssessment', 'MemoryComplaints', 'Beh
avioralProblems', 'ADL', 'Confusion', 'Disorientation', 'PersonalityChanges', 'DifficultyCompletingTasks', 'Forgetfulness']

3. Categorical Columns:
[]
```

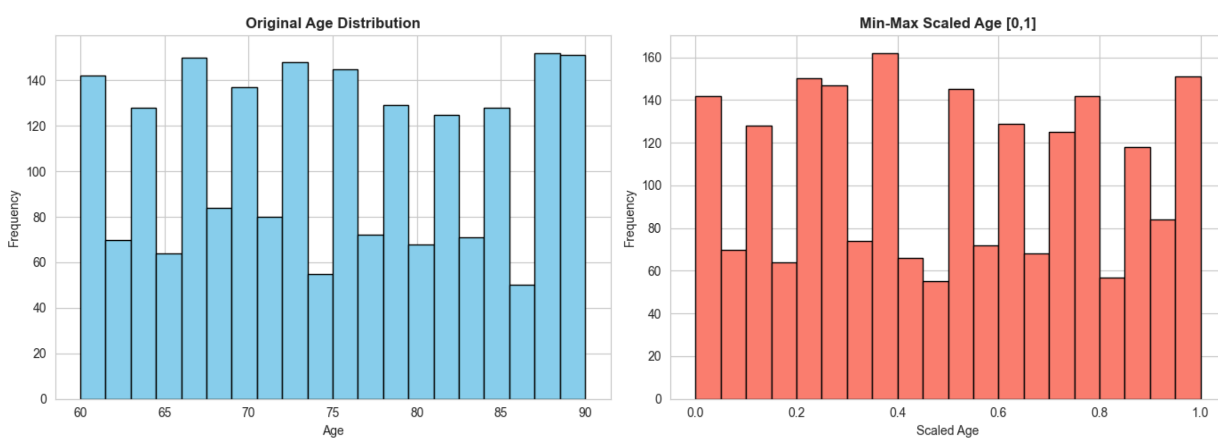
This step verifies and formats the dataset by inspecting data types and separating numerical and categorical features. Ensuring correct data types is essential before applying normalization, binning, and encoding, as machine learning algorithms require numerical and consistently formatted inputs.

- Data Normalization

```python
# DATA NORMALIZATION
print("\n--- Data Normalization ---")

# Import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

print("\n1. Min-Max Scaling (Normalization)")
print("   Formula: x_scaled = (x - x_min) / (x_max - x_min)")
print("   Result: Values scaled to range [0, 1]")

if 'Age' in X.columns:
    # Create a copy for demonstration
    min_max_scaler = MinMaxScaler()
    age_minmax = min_max_scaler.fit_transform(X[['Age']])

    print("\n   Example: Age Column")
    print(f"   Original range: [{X['Age'].min():.2f}, { (variable) age_minmax: ndarray[_AnyShape, dtype[Any]]
    print(f"   Scaled range: [{age_minmax.min():.2f}, {age_minmax.max():.2f}]")
```

```python
    # Show comparison
    comparison_df = pd.DataFrame({
        'Original_Age': X['Age'].head(10),
        'MinMax_Scaled': age_minmax[:10].flatten()
    })
    print("\n   Sample comparison:")
    print(comparison_df)

    # Visualize
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))
    axes[0].hist(X['Age'], bins=20, color='skyblue', edgecolor='black')
    axes[0].set_title('Original Age Distribution', fontsize=12, fontweight='bold')
    axes[0].set_xlabel('Age')
    axes[0].set_ylabel('Frequency')

    axes[1].hist(age_minmax, bins=20, color='salmon', edgecolor='black')
    axes[1].set_title('Min-Max Scaled Age [0,1]', fontsize=12, fontweight='bold')
    axes[1].set_xlabel('Scaled Age')
    axes[1].set_ylabel('Frequency')

    plt.tight_layout()
    plt.savefig('minmax_scaling.png', dpi=300)
    plt.show()

print("\n2. Z-Score Normalization (Standardization)")
print("   Formula: z = (x - μ) / σ")
print("   Note: This will be applied later using StandardScaler")
```

```
--- Data Normalization ---

1. Min-Max Scaling (Normalization)
   Formula: x_scaled = (x - x_min) / (x_max - x_min)
   Result: Values scaled to range [0, 1]

   Example: Age Column
   Original range: [60.00, 90.00]
   Scaled range: [0.00, 1.00]

   Sample comparison:
   Original_Age  MinMax_Scaled
0            73       0.433333
1            89       0.966667
2            73       0.433333
3            74       0.466667
4            89       0.966667
5            86       0.866667
6            68       0.266667
7            75       0.500000
8            72       0.400000
9            87       0.900000

2. Z-Score Normalization (Standardization)
   Formula: z = (x - μ) / σ
   Note: This will be applied later using StandardScaler
```

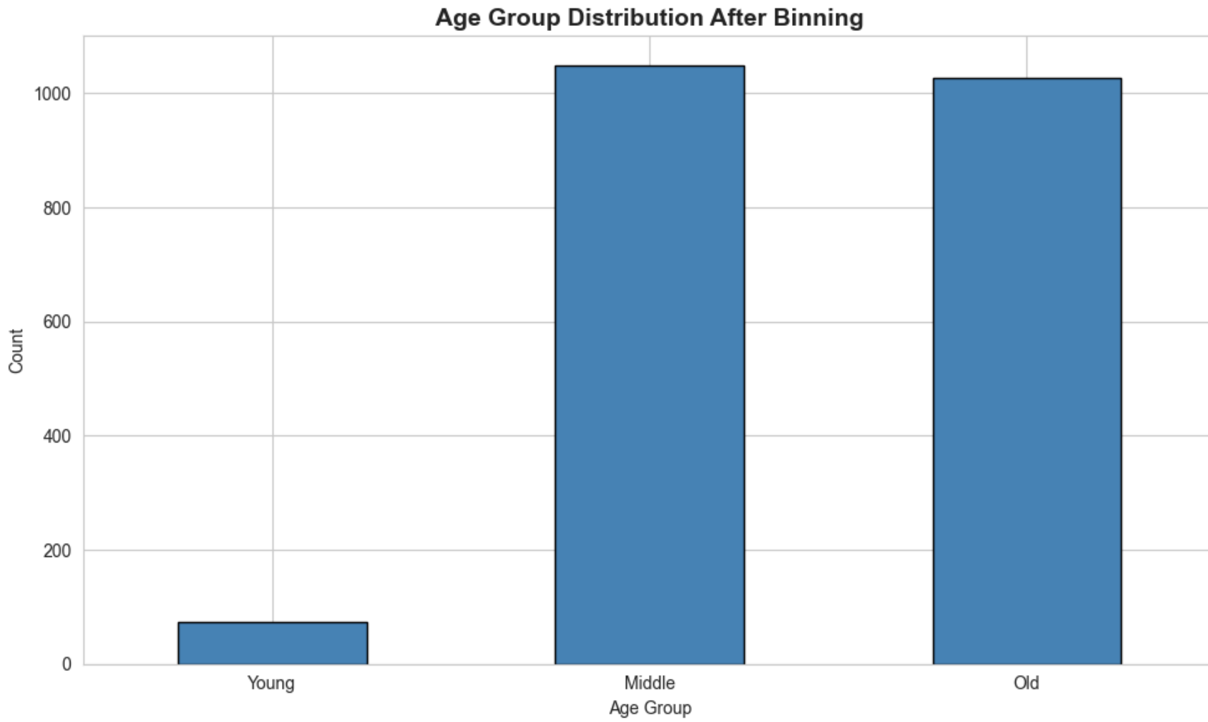


This output demonstrates Min–Max normalization applied to the Age feature, scaling values into the range [0,1]. The comparison between original and scaled values highlights how normalization prevents features with larger numerical ranges from dominating the learning process of distance-based and gradient-based models.

- Binning

```python
# BINNING
print("\n--- Binning ---")

if 'Age' in X.columns:
    print("\n1. Age Binning:")
    X['Age_binned'] = pd.cut(
        X['Age'],
        bins=[0, 60, 75, 100],
        labels=['Young', 'Middle', 'Old']
    )

    # Show binning results
    print("\n   Sample of Age Binning:")
    print(X[['Age', 'Age_binned']].head(10))

    print("\n   Age Group Distribution:")
    print(X['Age_binned'].value_counts())

    # Visualize
    plt.figure(figsize=(10, 6))
    X['Age_binned'].value_counts().sort_index().plot(kind='bar', color='steelblue', edgecolor='black')
    plt.title('Age Group Distribution After Binning', fontsize=14, fontweight='bold')
    plt.xlabel('Age Group')
    plt.ylabel('Count')
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.savefig('age_binning.png', dpi=300)
    plt.show()
```

```
--- Binning ---

1. Age Binning:

   Sample of Age Binning:
   Age Age_binned
0   73     Middle
1   89        Old
2   73     Middle
3   74     Middle
4   89        Old
5   86        Old
6   68     Middle
7   75     Middle
8   72     Middle
9   87        Old

   Age Group Distribution:
Age_binned
Middle    1048
Old       1027
Young       74
Name: count, dtype: int64
```



-

In this step, the continuous Age feature was transformed into categorical age groups using binning. This approach helps capture non-linear patterns related to age and Alzheimer's risk, improves interpretability, and allows the model to learn differences between broader age groups rather than relying only on exact numerical values.

- Indicator variables

```python
# INDICATOR VARIABLES
print("\n--- Indicator Variables (One-Hot Encoding) ---")

categorical_cols = X.select_dtypes(include=['object', 'category']).columns

if len(categorical_cols) > 0:
    print(f"\n1. Categorical columns to encode: {list(categorical_cols)}")

    # Show before encoding
    print("\n2. Before encoding (first 5 rows):")
    print(X[categorical_cols].head())

    # Apply one-hot encoding
    X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

    print(f"\n3. After one-hot encoding:")
    print(f"   Original features: {len(categorical_cols)}")
    print(f"   New indicator columns created: {len([col for col in X.columns if any(cat in col for cat in categorical_cols)])}")
    print(f"   Total columns now: {X.shape[1]}")
```

```
--- Indicator Variables (One-Hot Encoding) ---

1. Categorical columns to encode: ['Age_binned']

2. Before encoding (first 5 rows):
  Age_binned
0     Middle
1        Old
2     Middle
3     Middle
4        Old

3. After one-hot encoding:
   Original features: 1
   New indicator columns created: 2
   Total columns now: 30
```

These screenshots show the conversion of categorical features created through binning into numerical indicator variables using one-hot encoding. This transformation is necessary because machine learning models require numerical inputs and cannot directly process categorical text labels.

- Train–Test Split and Z-Score Normalization

```python

    # SPLIT AND SCALE
    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    # Scale the features (Z-score normalization)
    print("\n--- Final Scaling (Z-Score Normalization) ---")
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print(f"\nTraining set size: {X_train_scaled.shape}")
    print(f"Test set size: {X_test_scaled.shape}")

    print("\n" + "="*80)
    print("DATA PREPROCESSING COMPLETE")
    print("="*80)

    return X_train_scaled, X_test_scaled, y_train, y_test, X.columns

```

```
--- Final Scaling (Z-Score Normalization) ---

Training set size: (1719, 30)
Test set size: (430, 30)


================================================================================
DATA PREPROCESSING COMPLETE
================================================================================
```

The dataset was split into training and testing sets using stratified sampling to preserve class distribution. Z-score normalization was then applied using StandardScaler to standardize features, ensuring that all variables contribute equally during model training.